Incompressible Navier-Stokes with Particles Algorithm Design Document

Dan Martin and Phil Colella
Applied Numerical Algorithms Group

June 7, 2004

1 Overview

We would like to solve the incompressible Navier-Stokes equations with forces imposed on the fluid by suspended particles:

$$\begin{aligned} \frac{\partial \vec{u}}{\partial t} + (\vec{u} \cdot \nabla)\vec{u} &= -\nabla p + \nu \Delta \vec{u} + \vec{f} \\ \nabla \cdot \vec{u} &= 0 \end{aligned} \tag{1}$$

where $\vec{f}$ is the force on the fluid from the particles. We will use a drag-law type of relationship:

$$\vec{f}(\boldsymbol{x}) = \sum_k \vec{f}^{(k)} \delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k)}) \tag{2}$$

$$\frac{\partial \boldsymbol{x}^{(k)}}{\partial t} = \vec{u}^{(k)}(t) \tag{3}$$

$$\frac{\partial \vec{u}^{(k)}}{\partial t} = k(\vec{u}(\boldsymbol{x}^{(k)}(t), t) - \vec{u}^{(k)}(t)) = -\vec{f}^{(k)}, \tag{4}$$

where the sum is over the suspended particles, and $\delta_\epsilon(\boldsymbol{x})$ is a numerical (smoothed) approximation to the Dirac delta function $\delta(x)$:

$$\begin{aligned} \delta_\epsilon &= \frac{1}{2^{D-1}\pi\epsilon^D} g(\frac{r}{\epsilon}) \\ g(r) &\geq 0 \\ g(r) &= 0 \quad \text{for} \quad r > 1 \\ \int_0^1 g\ r^{D-1} dr &= 1 \end{aligned} \tag{5}$$

We need some way of getting the divergence-free contribution of the particle-induced forces. There are several options.

- (Straightforward approach, based on Peskin [5])
  Evaluate the force at all grid points, then project.

$$\vec{f_{\boldsymbol{i}}} = \vec{f}(\boldsymbol{i}h) \tag{6}$$

$$\mathcal{P}^h(f^h) \approx (\mathcal{P}\vec{f})(\boldsymbol{i}h) \tag{7}$$

  While this is a straightforward application of the immersed boundary approach for this problem (and is relatively simple to implement), the problem is that the force $\vec{f}$ is singular, so taking the derivatives necessary for the projection method is problematic from an accuracy standpoint.

  However, we do know that the projection of the singular forcing, $\mathcal{P}(\vec{f})$, is much less singular than $\vec{f}$ itself. As $\epsilon \to 0$, $\vec{f}$ is singular, while $\mathcal{P}(\vec{f})$ is continuous along the direction of $\vec{f}$.

- (Analytic approach)
  A second approach is to analytically determine the projection of the discrete delta function used to spread the particle force onto the mesh. If the projection operator is $(\boldsymbol{I} - grad(\Delta^{-1})div)$, then we can define $\boldsymbol{K}_\epsilon = \{K_{ij}\}$:

$$\boldsymbol{K}_\epsilon(\boldsymbol{x}) = (\boldsymbol{I} - grad(\Delta^{-1})div)\delta_\epsilon \tag{8}$$

  Ignoring the effects of physical boundaries, the operators can commute:

$$\boldsymbol{K}_\epsilon(\boldsymbol{x}) = \delta_\epsilon \boldsymbol{I} - grad\ div(\Delta^{-1})\delta_\epsilon \tag{9}$$

Note that $\Delta^{-1}\delta_\epsilon$ is a 1-D radial Poisson problem, which can be solved analytically (with the proper choice of δ_ϵ, of course).

Then, we can evaluate the projection of the forces on the grid:

$$\mathcal{P}\vec{f} = \sum_k \vec{f}^{(k)} \boldsymbol{K}_\epsilon(\boldsymbol{x} - \boldsymbol{x^{(k)}}) \tag{10}$$

While this approach avoids the singularity issues of the first approach, its problem is expense. Since $\boldsymbol{K}_\epsilon$ does not have compact support, the cost of this approach is $O(N_p N_g)$, where N_p is the number of particles, and N_g is the number of grid points.

- hybrid approach (based on the MLC algorithm of Cortez and Minion [2]).
  Take the Laplacian of (8):

$$\Delta \boldsymbol{K}_\epsilon = \Delta\delta_\epsilon - grad\ div(\delta_\epsilon \boldsymbol{I}) \tag{11}$$

Note that this *does* have compact support, since $\delta_\epsilon = 0$ for $r > \epsilon$.

Now define:

$$\boldsymbol{D_i^{(k)}} = \Delta^h \vec{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \tag{12}$$

where the $(\cdot - \boldsymbol{x}^{(k)})$ signifies evaluation at grid points, i.e. $(\boldsymbol{i}h - \boldsymbol{x}^{(k)})$. Then,

$$(\Delta^h)^{-1}\boldsymbol{D}^{(k)} = \vec{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \tag{13}$$

We can use the compact support of δ_ϵ and the harmonic nature of ** to make this easier. First, we can only evaluate D locally:

$$\boldsymbol{D_i^{(k)}} = \Delta^h \vec{f}^{(k)} \boldsymbol{K}_\epsilon(\cdot - \boldsymbol{x}^{(k)}) \ \text{ for } |\boldsymbol{i}h - \boldsymbol{x}^{(k)}| < (\epsilon + Ch) \tag{14}$$

where C is a safety factor. Then,

$$D_{\boldsymbol{i}} = \sum_k D_{\boldsymbol{i}}^{(k)} \tag{15}$$

$$\mathcal{P}\vec{f}(\boldsymbol{i}h) \approx (\Delta^h)^{-1}\boldsymbol{D} \tag{16}$$

If C is large enough, and if Δ^h is accurate enough, then this approximation is reasonable.

Note that we are referring only to the approaches in [5, 2] for spreading the force due to the particles to the computational mesh and applying them to the fluid. We will not use the approaches used to compute particle motion in [5, 2], since the particles in this work may move at a different velocity from the fluid, while in [5, 2], particles are constrained to move with the fluid.

2 Discretization of Advance

Using the third approach outlined above, we compute the projected force $\mathcal{P}_I(\vec{f}^n)$ using infinite-domain boundary conditions:

$$\mathcal{P}_I(\vec{f}^n)(\boldsymbol{x}) = (\Delta^h)^{-1} \sum_k \Delta^h \left(\vec{f}^{(k,n)} (\delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k,n)})) - grad\, div(\Delta^{-1}) \vec{f}^{(k,n)} \delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k,n)})) \right) \tag{17}$$

Then, we compute $\vec{u}^*$ in much the same way as the "normal" approach:

$$\vec{u}^* = \vec{u}^n - \Delta t[(\vec{u} \cdot \nabla)\vec{u}]^{n+\frac{1}{2}} - \Delta t G \pi^{n-\frac{1}{2}} + \Delta t[\nu \Delta \vec{u}] + \Delta t \mathcal{P}_I(\vec{f}^n) \tag{18}$$

where $[\nu \Delta \vec{u}]$ is computed using the TGA algorithm, and the advective term is computed using the second-order upwind scheme outlined in [4], including $\mathcal{P}_I(\vec{f}^n)$ as a force in the predictor step.

In order to make the time advance formally second-order in time, we now update the particle positions and velocities:

$$\vec{u}^{k,*} = \vec{u}^{k,n} + \Delta t \vec{f}^{k,n} \tag{19}$$

$$\boldsymbol{x}^{k,*} = \boldsymbol{x}^{k,n} + \Delta t \vec{u}^{k,n} \tag{20}$$

We then use the new particle data to compute the projected drag force (again using infinite-domain boundary conditions) $\mathcal{P}_I(\vec{f}^*)$. Then, modify $\vec{u}^*$ to make the update second-order in time and project:

$$\vec{u}^{n+1} = \mathcal{P}(\vec{u}^* + \Delta t[G\pi^{n-\frac{1}{2}} - \frac{1}{2}\mathcal{P}_I(\vec{f}^n) + \frac{1}{2}\mathcal{P}_I(\vec{f}^*)]) \tag{21}$$

Note that the projection is also applied to $\mathcal{P}_I(\vec{f})$; this is done to enforce the physical boundary conditions, since $\mathcal{P}_I(\vec{f})$ was computed using infinite-domain boundary conditions (with image particles used to approximate the effects of the solid walls for particles near the walls, as outlined in the following section).

We then can use the updated velocities to update the particle information to the new time level:

$$\vec{u}^{k,n+1} = \vec{u}^{k,n} + \frac{\Delta t}{2}(\vec{f}^{k,n} + \vec{f}^{k,*}) \tag{22}$$

$$\boldsymbol{x}^{k,n+1} = \boldsymbol{x}^{k,n} + \frac{\Delta t}{2}(\vec{u}^{k,n} + \vec{u}^{k,*}) \tag{23}$$

3 Evaluating $\mathcal{P}_I(\vec{f})$

We want to approximate the divergence-free contribution of the drag force $\mathcal{P}_I(\vec{f})$.

In indicial notation,

$$(\mathcal{P}\vec{f}(\boldsymbol{x}))_i = \sum_k f_j^{(k)}(\delta_{ij}\Delta - \partial_i\partial_j)(\Delta^{-1})\delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k)}) \tag{24}$$

Define $K_{ij}^{(k)}(\boldsymbol{x}) = (\delta_{ij}\Delta - \partial_i\partial_j)(\Delta^{-1})\delta_\epsilon(\boldsymbol{x} - \boldsymbol{x}^{(k)})$. Then,

$$\mathcal{P}f(\boldsymbol{x})_i = \sum_k f_j^{(k)} K_{ij}^{(k)} \tag{25}$$

Use a MLC-type approach; approximate $\boldsymbol{D} = \Delta\boldsymbol{K}$ by $\tilde{\boldsymbol{D}} = \{\tilde{D}_{ij}\}$:

$$\tilde{D}_{ij}^{(k)}(\boldsymbol{x}) = \begin{cases} \Delta^h K_{ij}^{(k)}(\boldsymbol{x}) & \text{if } \; r < \epsilon + Ch \\ 0 \;\; \text{otherwise} \end{cases} \tag{26}$$

Then,

$$\mathcal{P}f(\boldsymbol{x})_i = \sum_k f_j^{(k)} K_{ij}^{(k)}(\boldsymbol{x}) \tag{27}$$

$$\Delta^h\mathcal{P}f(\boldsymbol{x})_i = \sum_k f_j^{(k)} \Delta^h K_{ij}^{(k)}(\boldsymbol{x}) \tag{28}$$

$$\approx \sum_k f_j^{(k)} \tilde{D}_{ij}^{(k)}(\boldsymbol{x}) \tag{29}$$

$$\mathcal{P}f(\boldsymbol{x})_i \approx (\Delta^h)^{-1} \sum_k f_j^{(k)} \tilde{D}_{ij}^{(k)}(\boldsymbol{x}). \tag{30}$$

We solve (30) with infinite-domain boundary conditions on $\mathcal{P}_I\vec{f}(\boldsymbol{x})$ (note the subscript, which indicates the use of inifinite-domain boundary conditions

as opposed to the standard projection operator $\mathcal{P}(\vec{u})$, which uses regular physical boundary conditions).

To better approximate the no-normal-flow boundary condition at physical walls, we also use image particles for all particles near the wall. For each particle within $(ch + \epsilon)$ of the wall, we add an image particle on the other side of the wall with the opposite induced velocity field normal to the wall,

We create a much larger computational domain which contains our computational domain Ω, which we denote by Ω^*. We discret]ize Ω^* using a Cartesian mesh with a mesh spacing $H = O(h^{\frac{1}{2}})$. We transfer the problem to the larger and coarser domain Ω^* and solve on this larger domain:

$$\widetilde{\mathcal{P}_I f}(\boldsymbol{x})_i = (\Delta^H)^{-1} \sum_k f_j^{(k)} \tilde{D}_{ij}^{(k)}(\boldsymbol{x}) \quad \text{on} \quad \Omega^*, \tag{31}$$

using infinite-domain boundary conditions [1, 3].

3.1 Evaluating K

We need to compute $\mathbf{K} = \boldsymbol{I}\delta_\epsilon - grad\ div(\Delta^{-1})\delta_\epsilon$. First, define $W(r) = (\Delta^{-1})\delta_\epsilon$, and $Q(r) = \int_0^r \delta_\epsilon(s) s^{D-1} ds$. We will find it useful to evaluate $W(r)$ as follows:

$$\begin{aligned} \frac{1}{r^{D-1}} \frac{\partial}{\partial r} (r^{D-1} \frac{\partial}{\partial r} W(r)) &= \delta_\epsilon(r) \\ r^{D-1} \frac{\partial}{\partial r} W(r) &= \int_0^r \delta_\epsilon(s) s^{D-1} ds \\ \frac{\partial}{\partial r} W(r) &= \frac{Q(r)}{r^{D-1}} \end{aligned} \tag{32}$$

Then, we can evaluate $\boldsymbol{I}\delta_\epsilon - grad\ div(\Delta^{-1})\delta_\epsilon$:

$$K_{ij}^{(k)} = \delta_{ij}\delta_\epsilon(r) - \partial_i \partial j(\Delta^{-1})\delta_\epsilon(r) \tag{33}$$

$$= \delta_{ij}\delta_\epsilon(r) - \frac{\partial}{\partial x_i}(\frac{\partial}{\partial x_j} W(r)) \tag{34}$$

$$= \delta_{ij}\delta_\epsilon(r) - \frac{\partial}{\partial x_i}(\frac{\partial r}{\partial x_j}\frac{\partial}{\partial r} W(r)) \tag{35}$$

Without loss of generality, we use a coordinate system centered at $\boldsymbol{x}^{(k)}$, so that $(\boldsymbol{x} - \boldsymbol{x}^{(k)}) = \boldsymbol{x}$. We know that $\frac{\partial r}{\partial x_j} = \frac{x_j}{r}$. Then, using 32,

$$\begin{aligned} K_{ij}^{(k)} &= \delta_{ij}\delta_\epsilon(r) - \frac{\partial}{\partial x_i}\big((\frac{x_j}{r})\frac{Q(r)}{r^{D-1}}\big) \qquad (36) \\ &= \delta_{ij}\delta_\epsilon(r) - \big[\delta_{ij}\frac{Q(r)}{r^D} + \frac{x_i x_j}{r}\big(\frac{r^{D-1}\delta_\epsilon(r)}{r^D} - \frac{DQ(r)}{r^{D+1}}\big)\big] \\ K_{ij}^{(k)} &= \delta_{ij}\delta_\epsilon(r) - (\frac{\delta_{ij}}{r^D} - \frac{Dx_i x_j}{r^{D+2}})Q(r) - \frac{x_i x_j}{r^2}\delta_\epsilon(r) \qquad (37) \end{aligned}$$

4 Pseudocode description of approach

Here is an outline of the approach to solve for the divergence-free contribution of the drag force $\mathcal{P}_I(\vec{f})$:

1. For each space dimension, d, do:
   (a) set $D_{\boldsymbol{i},d}$ to 0 for each cell $\boldsymbol{i}$ in the domain.
   (b) For each cell $\boldsymbol{i}$ in the domain, do:
      i. Determine the group of cells $\boldsymbol{p}$ which fall within the region $|\boldsymbol{p}h - \boldsymbol{x}^{(k)}| \leq (Ch + \epsilon)$ for all particles k in cell $\boldsymbol{i}$.
      ii. For each particle k in cell $\boldsymbol{i}$ do:
         A. Using (37), compute $D_{\boldsymbol{p},d}^{(k)} = \Delta^h(f_j^{(k)} K_{dj}^{(k)})$ for all $\boldsymbol{p}$.
      iii. Increment D: $D_{\boldsymbol{p},d} := D_{\boldsymbol{p},d} + \sum_k D_{\boldsymbol{p},d}^{(k)}$
   (c) Solve for $\mathcal{P}_I(f_d)$: $\mathcal{P}_I(f_d) = (\Delta^h)^{-1} D_d$, using inifinite domain boundary conditions.

5 Specifying $\delta_\epsilon(r)$

To completely specify the numerical delta function δ_ϵ in (5), we need to specify the form of $g(r)$. For the initial implementation, we are taking g to be a polynomial function in r:

$$g(r) = A + Br + Cr^2 + Dr^3 + Er^4 + Fr^5 \qquad (38)$$

Two constraints on g are specified in (5); we supply 4 more. The full set of constraints are:

$$\begin{aligned} g(1) &= 0 \\ \int_0^1 g\, r^{D-1} dr &= 1 \\ g'(0) &= 0 \\ g'(1) &= 0 \\ g''(1) &= 0. \\ g'''(0) &= 0 \end{aligned} \tag{39}$$

The constraint on the third derivative term is designed to remove a singular term in the divergence which is proportional to g''' at the origin. Given these specifications, the constants in (38) for a 3 dimensional problem are:

$$\begin{aligned} A &= 21 \\ B &= 0 \\ C &= -70 \\ D &= 0 \\ E &= 105.0 \\ F &= -65 \end{aligned} \tag{40}$$

References

[1] Greg Balls. private communication, 2002.

[2] R. Cortez and M. Minion. The blob projection method for immersed boundary porblems. *JCP*, 161(2):428–453, July 2000.

[3] R. A. James. The solution of Poisson's equation for isolated source distributions. *JCP*, 25, 1977.

[4] D Martin and P Colella. A cell-centered adaptive projection method for the incompressible Euler equations. *J. Comput. Phys.*, 2000.

[5] C.S. Peskin and D.M. McQueen. A three-dimensional computational method for blood flow in the heart: I. immersed elastic fibers in a viscous incompressible fluid. *JCP*, 81:372, 1989.